```go
// Command-line arguments
// are a common way to parameterize execution of programs.

// For example, `go run hello.go` uses `run` and
// `hello.go` arguments to the `go` program.

package main

import  (
	"fmt"
	"os"
)

func main() {

	// `os.Args` provides access to raw command-line
	// arguments. Note that the first value in this slice
	// is the path to the program, and `os.Args[1:]`
	// holds the arguments to the program.
	fmt.Println("command-line arguments with program: \n", os.Args)
	fmt.Println("command-line arguments without program: \n", os.Args[1:])

	// You can get individual args with normal indexing.
	if len(os.Args) > 3  {
		arg := os.Args[3]
		fmt.Println(arg)
	}

	fmt.Print("command-line contains ", len(os.Args)-1, " argument(s) ")
	fmt.Println("/to say nothing of the path to the program/.")
	for n, arg := range os.Args  {
		fmt.Printf("%3d. %s\n", n, arg)
	}
}

/*
For such command-line:
  H:\Work. GO\command_line\command_line.exe a1 a2 a3 "a4 a41" a5

the program displays:
command-line arguments with program:
 [H:\Work. GO\command_line\command_line.exe a1 a2 a3 a4 a41 a5]
command-line arguments without program:
 [a1 a2 a3 a4 a41 a5]
a3
command-line contains 5 argument(s) /to say nothing of the path to the program/.
  0. H:\Work. GO\command_line\command_line.exe
  1. a1
  2. a2
  3. a3
  4. a4 a41
  5. a5
*/
```

```
57
58package main
59
60import (
61	"fmt"
62	"os"
63)
64
65func main() {
66	stat, err:= os.Stat("test.txt")
67	if err != nil {
68		fmt.Println(err)
69		return
70	}
71	fmt.Println(stat.Size())
72
73	f, err := os.OpenFile("test.1", os.O_APPEND, 0666)
74	if err != nil {
75		fmt.Println(err)
76		return
77	}
78
79	b:= []byte(" 012345")
80/*	эквивалентно
81	b:= []byte {32, 48, 49, 50, 51, 52}
82*/
83	n, err := f.Write(b)
84	fmt.Println(n)
85	if err != nil {
86		fmt.Println(err)
87		return
88	}
89
90	stat, err = os.Stat("test.1")
91	if err != nil {
92		fmt.Println(err)
93		return
94	}
95	fmt.Println(stat.Size())
96
97	err = f.Close()
98	if err != nil {
99		fmt.Println(err)
100		return
101	}
102	fmt.Println("file appended successfully")
103}
104
105
```

II_10d_append.go

```
106
107 package main
108 import (
109       "fmt"
110       "os"
111 )
112
113 func main() {
114       file, err := os.Open("morning.txt")
115       if err != nil {
116             fmt.Println(err)
117             return
118       }
119       defer file.Close()
120
121       fileinfo, err := file.Stat()
122       if err != nil {
123             fmt.Println(err)
124             return
125       }
126
127       filesize := fileinfo.Size()
128       buffer := make([]byte, filesize)
129       bytesread, err := file.Read(buffer)
130       if err != nil {
131             fmt.Println(err)
132             return
133       }
134
135       fmt.Println("bytes read: ", bytesread)
136       fmt.Println("bytestream to string: ")
137       fmt.Println(string(buffer))
138 }
139
```

II_10a_read1.go

```
140 package main
141 import (
142       "fmt"
143       "os"
144 )
145
146 func main() {
147       f, err := os.Create("test.txt")
148       if err != nil {
149             fmt.Println(err)
150             return
151       }
152       d2 := []byte{104, 101, 108, 108, 111, 32, 119, 111, 114, 108, 100}
153 //    d2 := []byte("hello world")  - то же самое
154       n2, err := f.Write(d2)
155       if err != nil {
156             fmt.Println(err)
157             f.Close()
158             return
159       }
160       fmt.Println(n2, "bytes written successfully")
161       err = f.Close()
162       if err != nil {
163             fmt.Println(err)
164             return
165       }
```

```
}
```
II_10c_write.go

```
package main

import (
	"fmt"
	"io"
	"os"
)

func main() {
	const BufferSize = 500
	file, err := os.Open("morning.txt")
	if err != nil {
		fmt.Println(err)
		return
	}
	defer file.Close()

	buffer := make([]byte, BufferSize)

	for {
		bytesread, err := file.Read(buffer)

		// err value can be io.EOF, which means that we reached the end of
		// file, and we have to terminate the loop. Note the fmt.Println lines
		// will get executed for the last chunk because the io.EOF gets
		// returned from the Read function only on the *next* iteration, and
		// the bytes returned will be 0 on that read.
		if err != nil {
			if err != io.EOF {
				fmt.Println(err)
			}

			break
		}

		fmt.Println("===> bytes read: ", bytesread)
		fmt.Println("===> bytestream to string: ")
		fmt.Println(string(buffer[:bytesread]))
	}
}

```
II_10b_read2.go